无向图连通性

2019年3月23日　　20:42

-
- Tarjan算法和无向图连通性

一. 割点和桥和Tarjan算法
   1. 割：删除以后使连通分支数增加的就是割点、割边/桥（完全图例外）
   2. Robert Tarjan的发明：线性求割法、证明并查集时间复杂度、斐波那契堆、Splay Tree、Lint-Cut Tree
   3. 时间戳：按先根dfs访问树结点的顺序，记为dfn
      i. 每次刚开启dfs(int x)，先给dfn[++tot] =x，便可得到连通分支中的dfn
      ii. 若在dfs函数体最前最后分别记录一次a[++tot]=x，得到的a中，每个结点下标x各出现两次，且两次x中间的序列是以x为根的子树
   4. 搜索树：在无向连通图中以任一结点为根，遍历每点各一次的dfs形成的树
      i. 即每次搜索到指向未知结点的边都加入树，否则忽视
      ii. 为了方便，之后管搜索树上的边叫搜索边
      iii. 无向不连通图也可用这种方法构成搜索森林
      iv. 无向图桥边，一定会出现在搜索树中
   5. 追溯值low[x]：x的以下相关结点的最小时间戳dfn
      i. 以x为根的搜索子树中的结点
      ii. 能从搜索树外的一条边到达“以x为根的搜索子树中的结点”的结点
   6. 追溯值求法
      i. 初值low[x]=dfn[x]
      ii. （扫描每条边(x,y)，若发现相邻点y无dfn，则选y做为搜索子树中x的子结点，选(x,y)为搜索子树上的边，对y做dfs，计算好y搜索子树的low值，再做iii；若y已有dfn，则(x,y)不应是搜索树的边，跳转iv）
      iii. 对搜索边(x,y)，low[x]=min(low[x],low[y])（y是刚dfs好的）
      iv. 对非搜索边(x,y)，low[x]=min(low[x],dfn[y])（且y不是x的父结点）
      v. 总结，指向子结点的搜索边，用low更新；不是搜索边，用dfn更新；是指向父结点的搜索边，就跳过（若指向父节点有多条边，则至少有一条不是搜索树的边，那就会用dfn更新）
   7. 割边判定：设y在x搜索子树上，无向边(x,y)是桥 iff dfn[x]<low[y]
      i. 因为满足后件时，说明y只能通过搜索树上的边(x,y)回到x（若有其他边使y能到达x或x的父节点z，则low[y]会更新到dfn[x]或dfn[z]）
      ii. 在常规的静态数组法加边时，跳过编号1，使23、45、67成对出现，就能通过编号xor1来找到反向边编号，并区分父子间平行边和反向边

```
int n,m,num,tot=1;  //点数，边数，点号，边号
int dfn[MN],low[MN],brdg[MN<<1];
int fst[MN],to[MN<<1],nxt[MN<<1];
inline void add(int x,int y){
    to[++tot] =y, nxt[tot] =fst[x], fst[x] =tot;
}
```

```
void tarjan(int x,int idx){      //从idx号边进入x结点，dfs
    low[x] = dfn[x] = ++num;
    for(int i= fst[x]; i; i= nxt[i]){
        int y= to[i];
        if(!dfn[y]){ //先dfs求好y的low
            tarjan(y, i);//选i作为搜索子树的边
            low[x]= min(low[x], low[y]);
            if(low[y] > dfn[x]) //是桥
                brdg[i] = brdg[i^1] = 1;
        }else if(i != (idx^1))     //不是搜索树父子边的反向边
            low[x]= min(low[x], dfn[y]);     //易知不在搜索树
    }
}

for__(i,1,m)
    scanf("%d%d",&x,&y),
    add(x,y),
    add(y,x);
for__(i,1,n)
    if(!dfn[i])  //新连通分支
        tarjan(i,0);
for(int i=2; i<tot; i+=2)
    if(brdg[i])
        printf("(%d,%d)是割边\n",to[i^1],to[i]);
```

8. 割点判定：非根结点x是割点 iff 搜索树上存在子结点y使dfn[x]<=low[x]
    i. 而根结点是需要两个不同子结点y同时满足后件时才是割点
    ii. 因为x的搜索子树的结点访问不到x的父节点时，删除x就能让子树和父节点断开，因此即使用搜索树的边去更新low[y]=dfn[x]也不影响判断
    iii. 应当把自环去了，防止错误判断

```
int n,m,num,tot=1;  //点数，边数，点号，边号
int dfn[MN],low[MN],sz[MN],cut[MN];
int fst[MN],to[MN<<1],nxt[MN<<1];
int root;     //临时存储当前连通分支的搜索树根
void tarjan(int x){ //深搜x结点
    low[x] = dfn[x] = ++num;
    //sz[x] = 1;
    int flag =0; //专用于计数根结点有几棵搜索子树
    for(int i= fst[x]; i; i= nxt[i]){
        int y= to[i];
        if(!dfn[y]){ //先dfs求好y的low
            tarjan(y);
            //sz[x] += sz[y];
            low[x]= min(low[x], low[y]);
            if(low[y] >= dfn[x]){      //找到y使x可能是桥
                if(x!=root || flag) //非根 或 是根且有两个y
                    cut[x] = 1;
                ++flag;
            }
        }else  //不用管是不是反向边，反正取等也是割点
            low[x]= min(low[x], dfn[y]);
    }
}

for__(i,1,m){
```

```
        scanf("%d%d",&x,&y);
        if(x==y)      //忽视自环
            continue;
        add(x,y),
        add(y,x);
    }
    for__(i,1,n)
        if(!dfn[i])  //新连通分支
            root = i,     //全局变量存储当前dfs的连通分支的根
            tarjan(i);
    for__(i,1,n)
        if(cut[i])
            printf("%d是割点\n",i);
```

二. double connected component双连通分量DCC

1. 双连通图：不存在割点的是点双连通图；不存在桥的是边双连通图
2. 双连通分量：极大点双连通子图是点双连通分量VDCC；极大边双连通子图是边双连通分量EDCC，统称为双连通分量DCC
3. 无割定理：
    i. |V|>=2的无向连通图，是VDCC iff 任两点都共环（除起点无重复点环）
    ii. 证明：任两点共环 iff 任两点有两种不相交通路 iff 无割点
    iii. |E|>=2的无向连通图，是EDCC iff 任一边都处于环（同上的环）
    iv. 证明：任一边处于环 iff 删任一边都能保持环上其他点连通 iff 无割边
4. EDCC求法：忽略桥，深搜

```
    int e[MN],ecnt;      //e[x]= x所属edcc编号，ecnt= 找到的edcc数量
    void edfs(int x){    //深搜找当前x所在edcc的所有点
        e[x] = ecnt;
        for(int i= fst[x]; i; i= nxt[i])
            if(!brdg[i] && !e[to[i]])
                edfs(to[i]);
    }

    for__(x,1,n)
        if(!e[x])
            ++ecnt,
            edfs(x);
```

5. EDCC缩点：遍历每条边，若所连点在不同EDCC中则给这两个EDCC编号加边

```
    int efst[MN],eto[MN<<1],enxt[MN<<1],etot=1;
    void eadd(int x,int y){
        eto[++etot] =y, enxt[etot] =efst[x], efst[x] =etot;
    }
    for__(i,2,tot){
        int x= to[i];
        int y= to[i^1];
        if(e[x]!=e[y])
            eadd(e[x],e[y]);
    }
```

6. VDCC求法
    i. 首先要知道什么是VDCC：
        1) 孤立点是VDCC
        2) 两点一边且“极大”的子图也是VDCC
        3) 割点可以出现在多个VDCC的角落（而桥不会出现在EDCC）
    ii. 求法简介：

1) 全局维护一个栈，每次新访问结点（打dfn）时给结点编号入栈
2) 特判一下如果当前编号的点是孤立点，则自成一个VDCC，return
3) 当找到dfn[x]<=low[y]（即使x是根，不确定是不是割点）时弹出栈顶所有结点进新的VDCC集合，直至找到y，再把X本身加入集合

7. VDCC缩点
   i. 因为割点会出现在不同VDCC，所以常见做法是先给各割点单独编号（紧跟着VDCC数量+1开始编），再给每个VDCC集合遍历，让VDCC与割点连边，让非割点直接归属于该VDCC编号
   ii. 即p个割点和t个VDCC的图会缩成p+t个点的图
      -
      - 例题

三. BLO（P3469）
1. 题意：无向连通图中删去各点的关联边后会产生多少不连通的有序点对
2. 引：求有序对，只需累加每个点为起点时有几个终点不可达
3. 非割点的边被删除后，该点无法连通其他n-1个点，其他n-1个点无法连通该点
4. 割点的边被删除后讨论3种情况：
   i. 割点x不可达其他n-1个点
   ii. low[y]>=dfn[x]的y搜索子树的sz[y]个点不可达n-sz[y]个点
   iii. 设以上情况共统计到了1+sum个点，则剩余n-1-sum个点不可达他们
5. 顺带一提这题靠删除关联边产生的不连通分量和VDCC或EDCC无关

四. educoder.net/tasks/cupygblie9fs
1. 题意：无向图缩点后求直径。两次dfs即可
2. 对比cupygblie9fs是有向图缩点，求最长路就不适合对所有零入度点做两次dfs了，但有向图可以用拓扑排序，先缩点加边时给y的入度++，给所有缩点图的零入度点入队，再：

```
while(q.size()){            //“求拓扑排序”顺便求DAG最长路
    int x= q.front();   q.pop();
    for(int i= sfst[x]; i; i= snxt[i]){
        int y= sto[i];
        if(dst[y] < dst[x] +sval[i])
            dst[y]= dst[x] +sval[i],
            ans= max(ans, dst[y]);
        if(--deg[y] ==0)
            q.push(y);}}
```